

# DCC004 - Algoritmos e Estruturas de Dados II

## Interfaces e Polimorfismo

---

Renato Martins

Email: [renato.martins@dcc.ufmg.br](mailto:renato.martins@dcc.ufmg.br)

<https://www.dcc.ufmg.br/~renato.martins/courses/DCC004>

Material adaptado de PDS2 - Douglas Macharet e Flávio Figueiredo

# Introdução

- Termo originário do grego
  - Poli: muitas
  - Morphos: formas
- POO
  - Objetos de classes diferentes responderem a uma mesma mensagem de diferentes maneiras
- Várias formas de responder à mensagem

# Introdução

- Utilizar um mesmo nome para se referir a diferentes métodos sobre um certo tipo
- Objeto decide qual método deve ser
- Exemplo
  - Hierarquia de mensagens
  - Classe mais genérica possui o método **exibir**

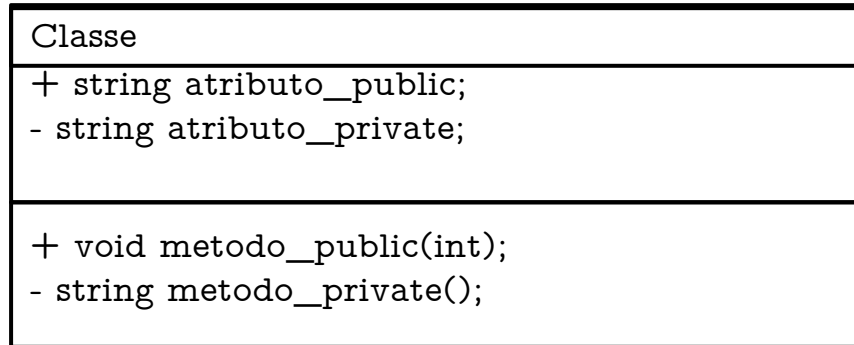
# Interfaces

---

# Diferentes tipos de Mensagens

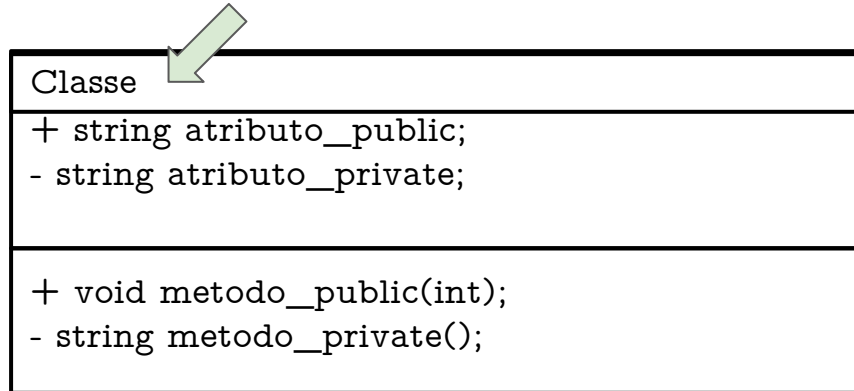


# Unified Modelling Language



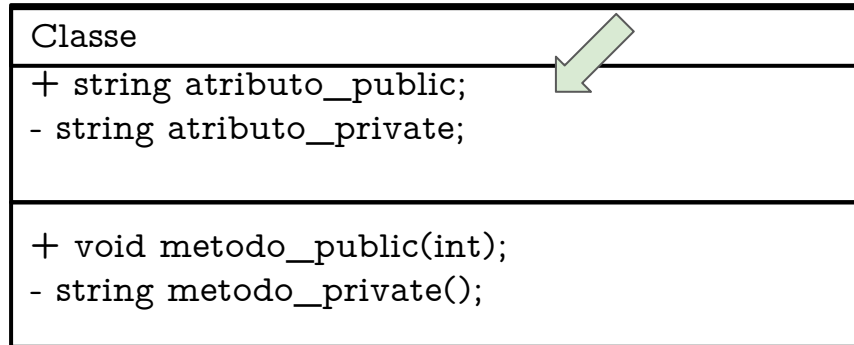
- UML define uma padrão de diagramas
- Úteis para o resto da disciplina

# Unified Modelling Language



Nome da Classe

# Unified Modelling Language



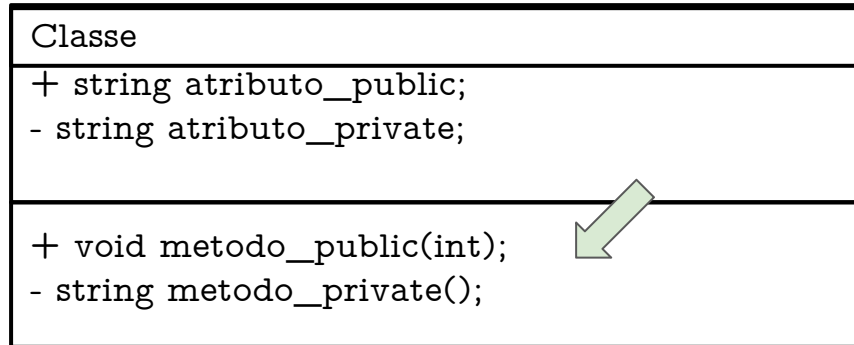
## Atributos

+ → public

- → private



# Unified Modelling Language

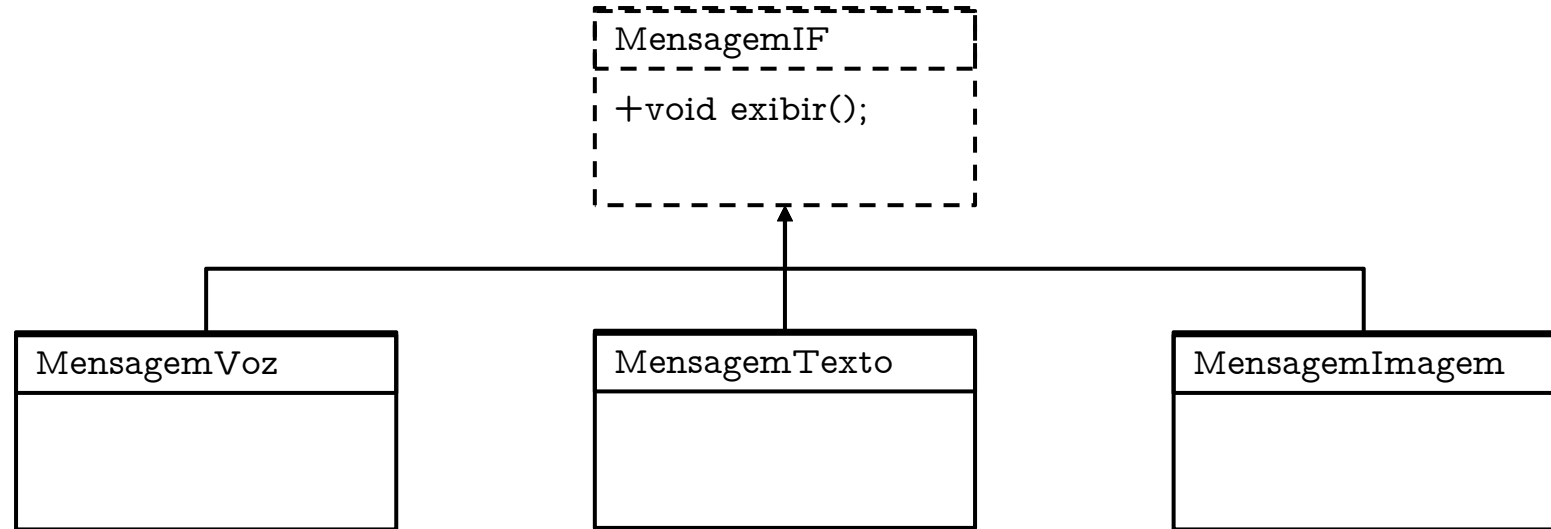


## Métodos

+ → public

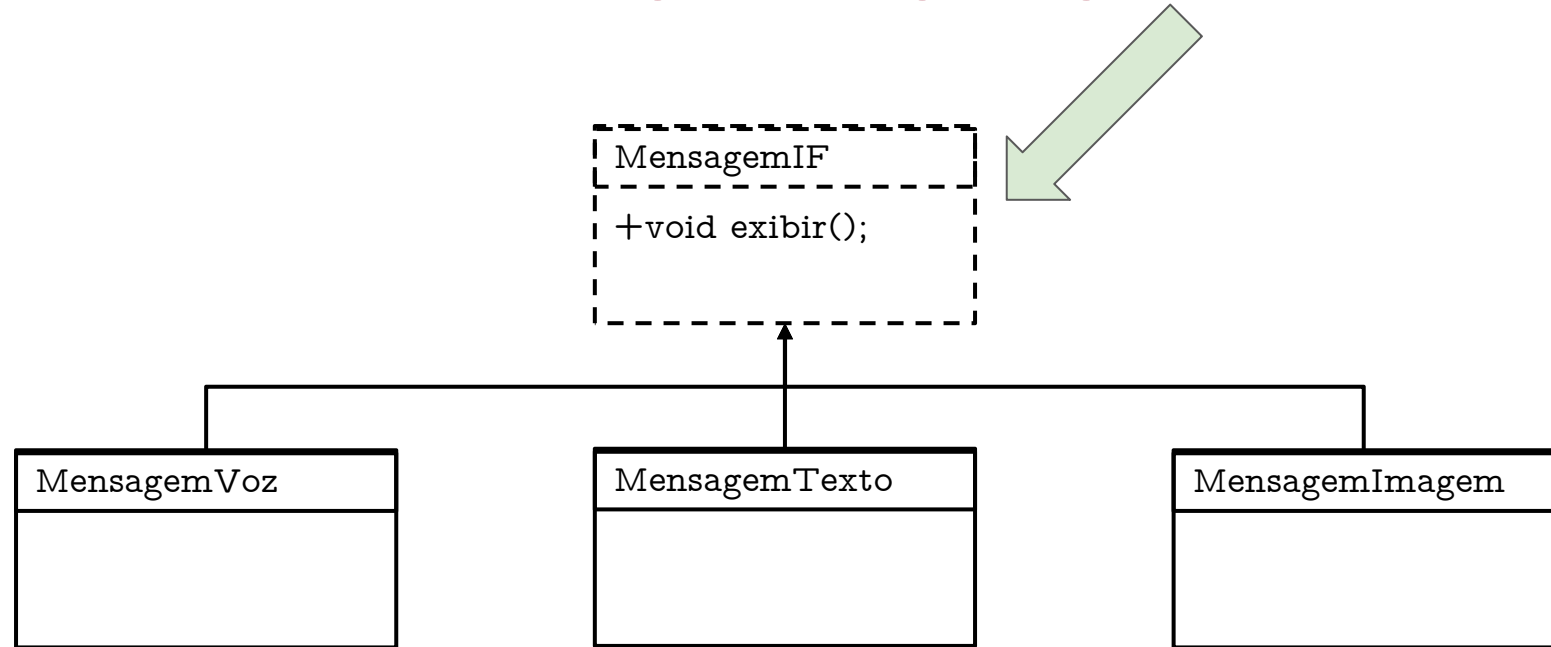
- → private

# Unified Modelling Language



Entendendo o diagrama:

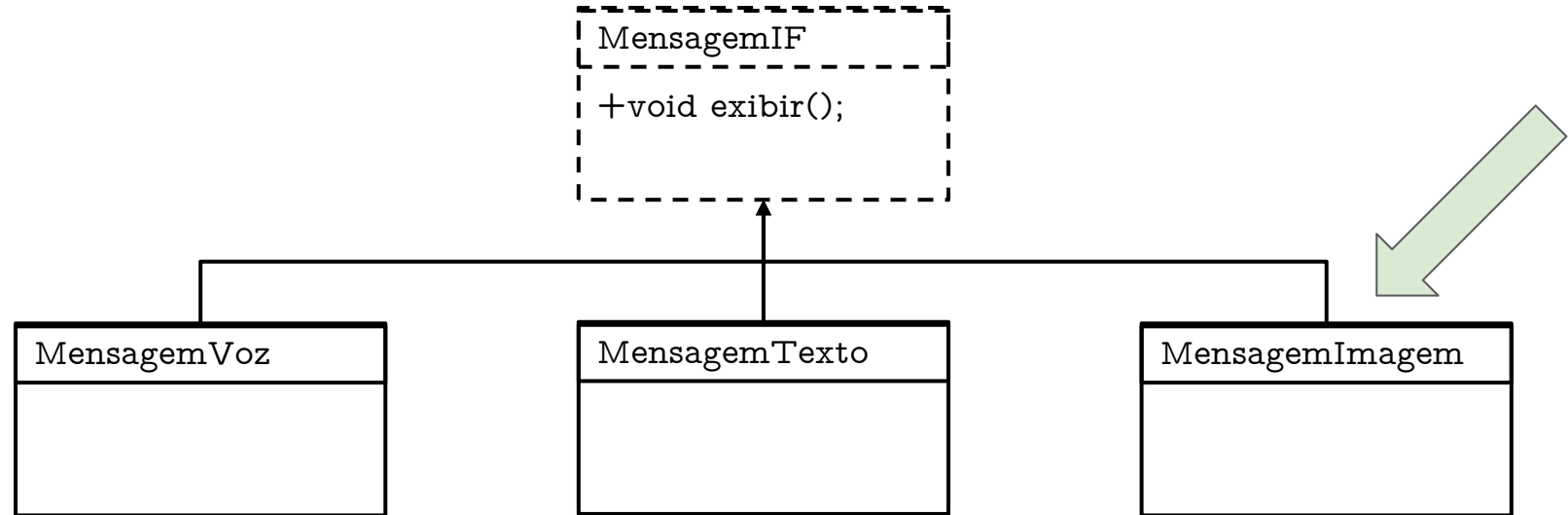
# Unified Modelling Language



## Interfaces

- Topo da hierarquia de mensagens
- Pontilhada pois nunca é implementada

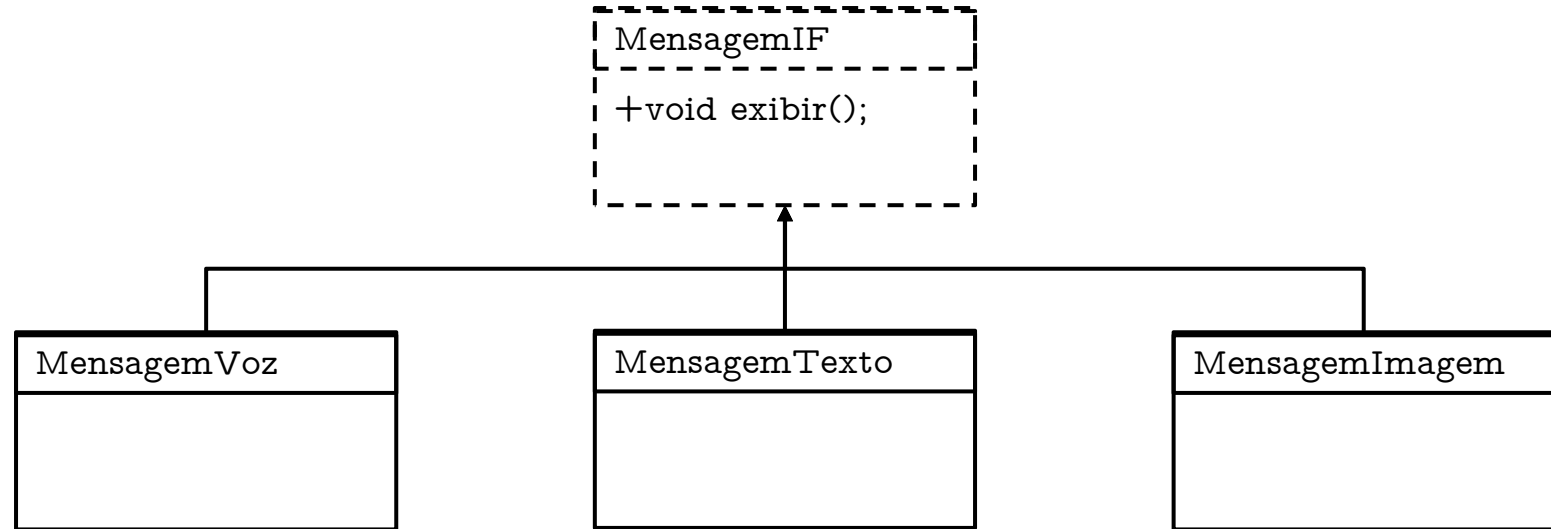
# Unified Modelling Language



Classes:

- Já conhecemos elas
- Definem um comportamento comum

# Comportamento Padrão



- Como esses objetos devem responder ao receberem o mesmo sinal: exibir?
  - Todos respondem da mesma forma?
- Vai existir um comportamento padrão?

# Polimorfismo

- Programação voltada a tipos abstratos
- Possibilidade de um tipo abstrato (classe abstrata ou interface) ser utilizado sem que se conheça a implementação concreta
  - Independência de implementação
  - Maior foco na interface (fronteira, contrato)

# Interfaces

- Definidas com métodos virtuais
- Não podem ser instanciadas
- virtual, = 0 garantem isso

```
#ifndef PDS2_MENSAGEM_H
#define PDS2_MENSAGEM_H

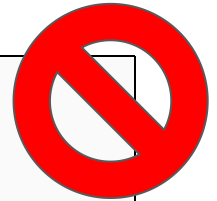
class MensagemIF {
public:
    virtual void exibir() = 0;
};

#endif
```

# Interfaces

- Não podem ser instanciadas
- 2 linhas com erro abaixo

```
int main(void) {  
    MensagemIF msg = MensagemIF();  
    MensagemIF *msg2 = new MensagemIF();  
    MensagemIF msg3;  
}
```

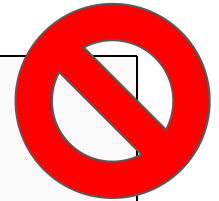




# Interfaces

- Como fazer uso?!

```
int main(void) {  
    MensagemIF msg = MensagemIF();  
    MensagemIF *msg2 = new MensagemIF();  
    MensagemIF msg3;  
}
```



# Implementando Interfaces

## Definimos o comportamento nas classes

```
#ifndef PDS2_MENSAGEMTEXTO_H
#define PDS2_MENSAGEMTEXTO_H

#include <string>
#include "mensagem.h"

class MensagemTexto : public MensagemIF {
private:
    std::string __msg;
public:
    MensagemTexto(std::string msg);
    virtual void exhibir();
};

#endif
```

# Implementando Interfaces

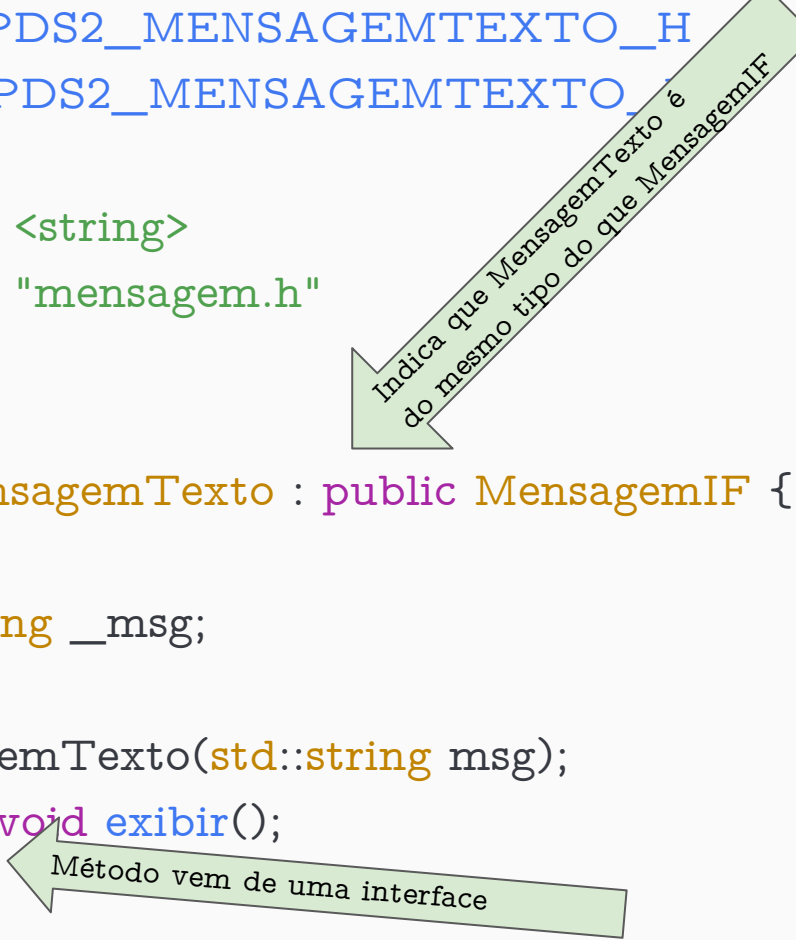
## Definimos o comportamento nas classes

```
#ifndef PDS2_MENSAGEMTEXTO_H
#define PDS2_MENSAGEMTEXTO_H

#include <string>
#include "mensagem.h"

class MensagemTexto : public MensagemIF {
private:
    std::string _msg;
public:
    MensagemTexto(std::string msg);
    virtual void exibir();
};

#endif
```



Indica que MensagemTexto e do mesmo tipo do que MensagemIF

Método vem de uma interface

# Implementando Interfaces

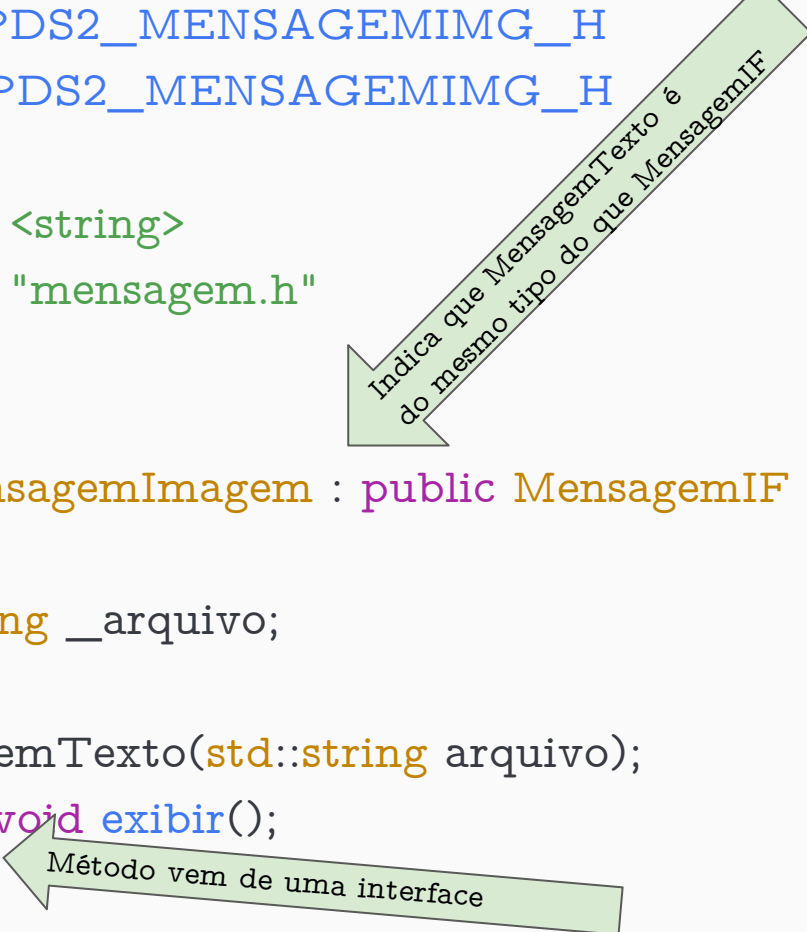
Podemos definir uma mensagem com imagens

```
#ifndef PDS2_MENSAGEMIMG_H
#define PDS2_MENSAGEMIMG_H

#include <string>
#include "mensagem.h"

class MensagemImagem : public MensagemIF {
private:
    std::string __arquivo;
public:
    MensagemTexto(std::string arquivo);
    virtual void exibir();
};

#endif
```



Indica que MensagemTexto e do mesmo tipo do que MensagemIF

Método vem de uma interface

# Implementando

## Mais de um tipo de mensagem

```
#include "mensagemtexto.h"

#include <iostream>

MensagemTexto::MensagemTexto(std::string msg) {
    this->_msg = msg;
}

void MensagemTexto::exibir() {
    std::cout << this->_msg;
    std::cout << std::endl;
}
```

```
#include "mensagemimg.h"

#include <fstream>
#include <iostream>

MensagemImagem::MensagemImagem(std::string arquivo) {
    this->_arquivo = arquivo;
}

void MensagemImagem::exibir() {
    std::ifstream arquivo(this->_arquivo);
    std::string line;
    while (std::getline(arquivo, line))
        std::cout << line << std::endl;
    arquivo.close();
}
```

# Implementando

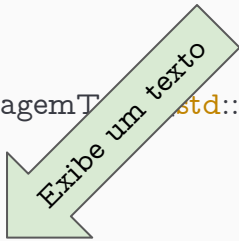
## Mais de um tipo de mensagem

```
#include "mensagemtexto.h"
```

```
#include <iostream>
```

```
MensagemTexto::MensagemTexto(std::string msg) {  
    this->_msg = msg;  
}
```

```
void MensagemTexto::exibir() {  
    std::cout << this->_msg;  
    std::cout << std::endl;  
}
```



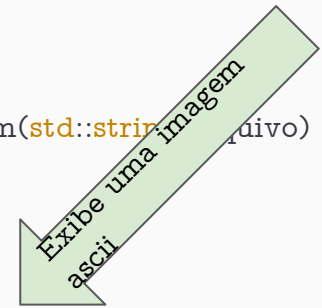
```
#include "mensagemimg.h"
```

```
#include <fstream>
```

```
#include <iostream>
```

```
MensagemImagem::MensagemImagem(std::string arquivo) {  
    this->_arquivo = arquivo;  
}
```

```
void MensagemImagem::exibir() {  
    std::ifstream arquivo(this->_arquivo);  
    std::string line;  
    while (std::getline(arquivo, line))  
        std::cout << line << std::endl;  
    arquivo.close();  
}
```



# Polimorfismo em ação

```
#include "mensagem.h"
#include "mensagemimg.h"
#include "mensagemtexto.h"
#include "mensagemvoz.h"

void exibir_na_tela(MensagemIF &msg) {
    msg.exibir();
}

int main(void) {
    MensagemTexto texto("Oi, tem aula de PDS2 hoje?");
    MensagemVoz audio("audio.wav");
    MensagemImagem image("imagem03.ascii");
    MensagemTexto texto2("Mas que puxa :)");

    exibir_na_tela(texto);
    exibir_na_tela(audio);
    exibir_na_tela(image);
    exibir_na_tela(texto2);
}
```

# Polimorfismo em ação

```
#include "mensagem.h"
#include "mensagemimg.h"
#include "mensagemtexto.h"
#include "mensagemvoz.h"

void exibir_na_tela(MensagemIF &msg) {
    msg.exibir();
}

int main(void) {
    MensagemTexto texto("Oi, tem aula de PDS2 hoje?");
    MensagemVoz audio("audio.wav");
    MensagemImagem image("imagem03.ascii");
    MensagemTexto texto2("Mas que puxa :)");

    exibir_na_tela(texto);
    exibir_na_tela(audio);
    exibir_na_tela(image);
    exibir_na_tela(texto2);
}
```

```
$ ./main
```



# Qual o tipo?!

```
#include "mensagem.h"
#include "mensagemimg.h"
#include "mensagemtexto.h"
#include "mensagemvoz.h"

void exibir_na_tela(MensagemIF &msg) {
    msg.exibir();
}

int main(void) {
    MensagemTexto texto("Oi, tem aula de PDS2 hoje?");
    MensagemVoz audio("audio.wav");
    MensagemImagem image("imagem03.ascii");
    MensagemTexto texto2("Mas que puxa :)");

    exibir_na_tela(texto);
    exibir_na_tela(audio);
    exibir_na_tela(image);
    exibir_na_tela(texto2);
}
```

```
$ ./main
```

# Qual o tipo?!

```
#include "mensagem.h"
#include "mensagemimg.h"
#include "mensagemtexto.h"
#include "mensagemvoz.h"

void exibir_na_tela(MensagemIF &msg) {
    msg.exibir();
}

int main(void) {
    MensagemTexto texto("Oi, tem aula de PDS2 hoje?");
    MensagemVoz audio("audio.wav");
    MensagemImagem image("imagem03.ascii");
    MensagemTexto texto2("Mas que puxa :)");

    exibir_na_tela(texto);
    exibir_na_tela(audio);
    exibir_na_tela(image);
    exibir_na_tela(texto2);
}
```

```
$ ./main
```

```
Oi, tem aula de PDS2 hoje?
```

# Qual o tipo?!

```
#include "mensagem.h"
#include "mensagemimg.h"
#include "mensagemtexto.h"
#include "mensagemvoz.h"

void exibir_na_tela(MensagemIF &msg) {
    msg.exibir();
}

int main(void) {
    MensagemTexto texto("Oi, tem aula de PDS2 hoje?");
    MensagemVoz audio("audio.wav");
    MensagemImagem image("imagem03.ascii");
    MensagemTexto texto2("Mas que puxa :)");

    exibir_na_tela(texto);
    exibir_na_tela(audio);
    exibir_na_tela(image);
    exibir_na_tela(texto2);
}
```

```
$ ./main
```

```
Oi, tem aula de PDS2 hoje?
```

# Qual o tipo?!

```
#include "mensagem.h"
#include "mensagemimg.h"
#include "mensagemtexto.h"
#include "mensagemvoz.h"

void exibir_na_tela(MensagemIF &msg) {
    msg.exibir();
}

int main(void) {
    MensagemTexto texto("Oi, tem aula de PDS2 hoje?");
    MensagemVoz audio("audio.wav");
    MensagemImagem image("imagem03.ascii");
    MensagemTexto texto2("Mas que puxa :)");

    exibir_na_tela(texto);
    exibir_na_tela(audio);
    exibir_na_tela(image);
    exibir_na_tela(texto2);
}
```

```
$ ./main
```

```
Oi, tem aula de PDS2 hoje?
```

```
Tocando o arquivo... audio.wav
```

# Qual o tipo?!

```
#include "mensagem.h"
#include "mensagemimg.h"
#include "mensagemtexto.h"
#include "mensagemvoz.h"

void exibir_na_tela(MensagemIF &msg) {
    msg.exibir();
}

int main(void) {
    MensagemTexto texto("Oi, tem aula de PDS2 hoje?");
    MensagemVoz audio("audio.wav");
    MensagemImagem image("imagem03.ascii");
    MensagemTexto texto2("Mas que puxa :)");

    exibir_na_tela(texto);
    exibir_na_tela(audio);
    exibir_na_tela(image);
    exibir_na_tela(texto2);
}
```

```
$ ./main
```

```
Oi, tem aula de PDS2 hoje?
```

```
Tocando o arquivo... audio.wav
```

# Qual o tipo?!

```
#include "mensagem.h"
#include "mensagemimg.h"
#include "mensagemtexto.h"
#include "mensagemvoz.h"

void exibir_na_tela(MensagemIF &msg) {
    msg.exibir();
}

int main(void) {
    MensagemTexto texto("Oi, tem aula de PDS2 hoje?");
    MensagemVoz audio("audio.wav");
    MensagemImagem image("imagem03.ascii");
    MensagemTexto texto2("Mas que puxa :)");

    exibir_na_tela(texto);
    exibir_na_tela(audio);
    exibir_na_tela(image);
    exibir_na_tela(texto2);
}
```

```
$ ./main
```

```
Oi, tem aula de PDS2 hoje?
```

```
Tocando o arquivo... audio.wav
```

# Qual o tipo?!

```
#include "mensagem.h"
#include "mensagemimg.h"
#include "mensagemtexto.h"
#include "mensagemvoz.h"

void exibir_na_tela(MensagemIF &msg) {
    msg.exibir();
}

int main(void) {
    MensagemTexto texto("Oi, tem aula de PDS2 hoje?");
    MensagemVoz audio("audio.wav");
    MensagemImagem image("imagem03.ascii");
    MensagemTexto texto2("Mas que puxa :)");

    exibir_na_tela(texto);
    exibir_na_tela(audio);
    exibir_na_tela(image);
    exibir_na_tela(texto2);
}
```

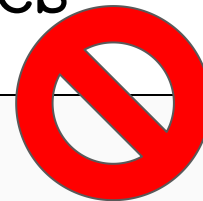
```
$ ./main
Oi, tem aula de PDS2 hoje?
Tocando o arquivo... audio.wav
```

```
      /      \
     : o      o ;
    (          ( )
     :          ;
    \      _      /
   - - - - -
   / ~~~~~ \
      /      \
     /|/\|/\|_ \|
    ( _|/\|/\|_\ )
                        |_____|
                       ___)_ |_(
    ( _____|_____ )
```

# Erros Comuns

- Tentar usar o tipo genérico na declaração
- Erro de compilação
- Tipos com tamanhos diferentes

```
void exibir_na_tela(MensagemIF &msg) {  
    msg.exibir();  
}  
  
int main(void) {  
    MensagemIF texto = MensagemTexto("Oi, tem aula de PDS2 hoje?");  
    MensagemIF audio = MensagemVoz("audio.wav");  
  
    exibir_na_tela(texto);  
    exibir_na_tela(audio);  
}
```





# Solução (se necessário)

- Ponteiros
- Sempre tem um tamanho fixo

```
#include "mensagem.h"
#include "mensagemtexto.h"
#include "mensagemvoz.h"

void exibir_na_tela(MensagemIF *msg) {
    msg->exibir();
}

int main(void) {
    MensagemIF *texto = new MensagemTexto("Oi, tem aula de PDS2 hoje?");
    MensagemIF *audio = new MensagemVoz("audio.wav");
    exibir_na_tela(texto);
    exibir_na_tela(audio);
    delete texto;
    delete audio;
}
```

# Tipos de Polimorfismo

---

# Polimorfismo

- Seleção da instância (forma) do objeto
  - Ligação Prematura (Early binding)
    - As decisões são feitas durante a compilação
  - Ligação Tardia (Late binding)
    - As decisões são feitas durante a execução
    - É a chave para o funcionamento do polimorfismo
- C++ ⇒ Padrão é ligação prematura
  - Ligação tardia utiliza o comando “virtual”

# Tipos de polimorfismo

## Early Binding

- Torna a linguagem mais expressiva
  - Templates em C++
- Universal paramétrico
  - Os tipos são identificados pelo compilador
  - São passados implicitamente à função

# Tipos de polimorfismo

## Early Binding


```
#include <list>

int main() {
    std::list<Pessoa> lista;

    Pessoa p;
    lista.push_back(p);

    std::cout << lista.size() << std::endl;

    return 0;
}
```



Template  
Universal  
Paramétrico

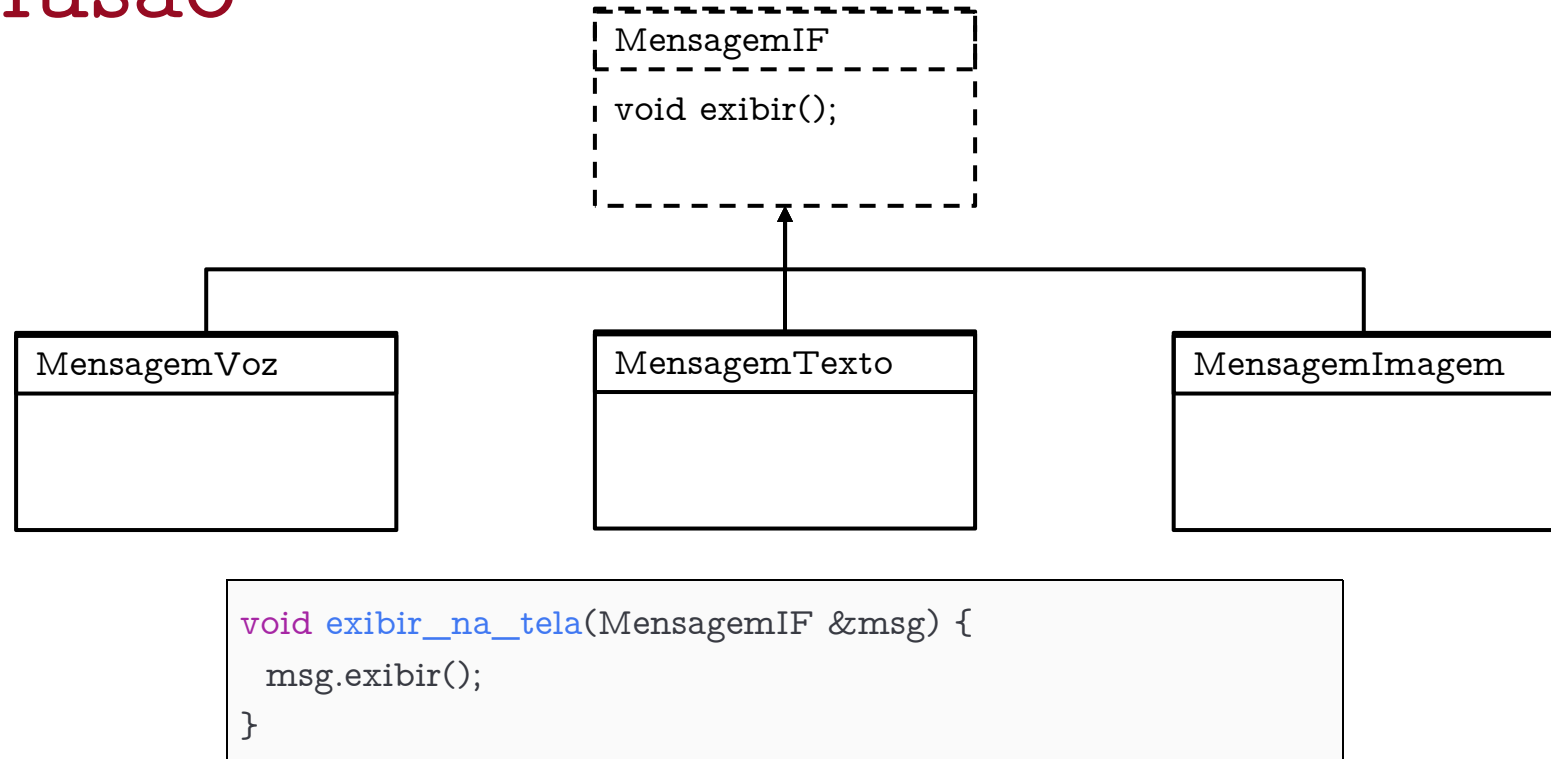
# Tipos de polimorfismo

## Late Binding

- Modela subtipos
  - Redefinição em classes descendentes
  - O subtipo está incluído no próprio tipo
- Onde um objeto de um tipo for esperado, um objeto do subtipo deve ser aceito
  - Princípio da substituição de Liskov
  - O contrário nem sempre é válido!

# Tipos de polimorfismo

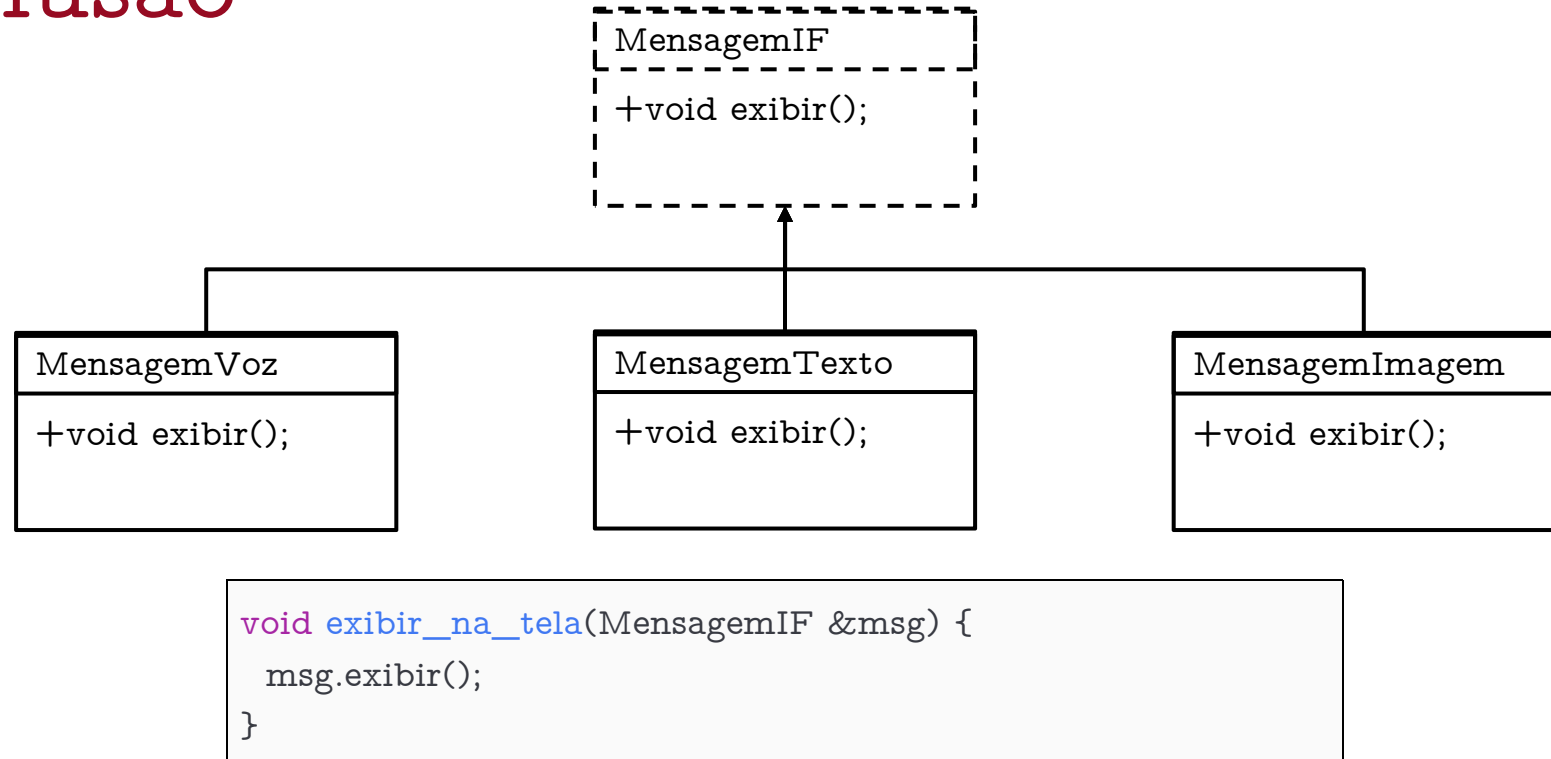
## Inclusão



- Podemos passar qualquer subtipo de `MensagemIF` para o método acima

# Tipos de polimorfismo

## Inclusão

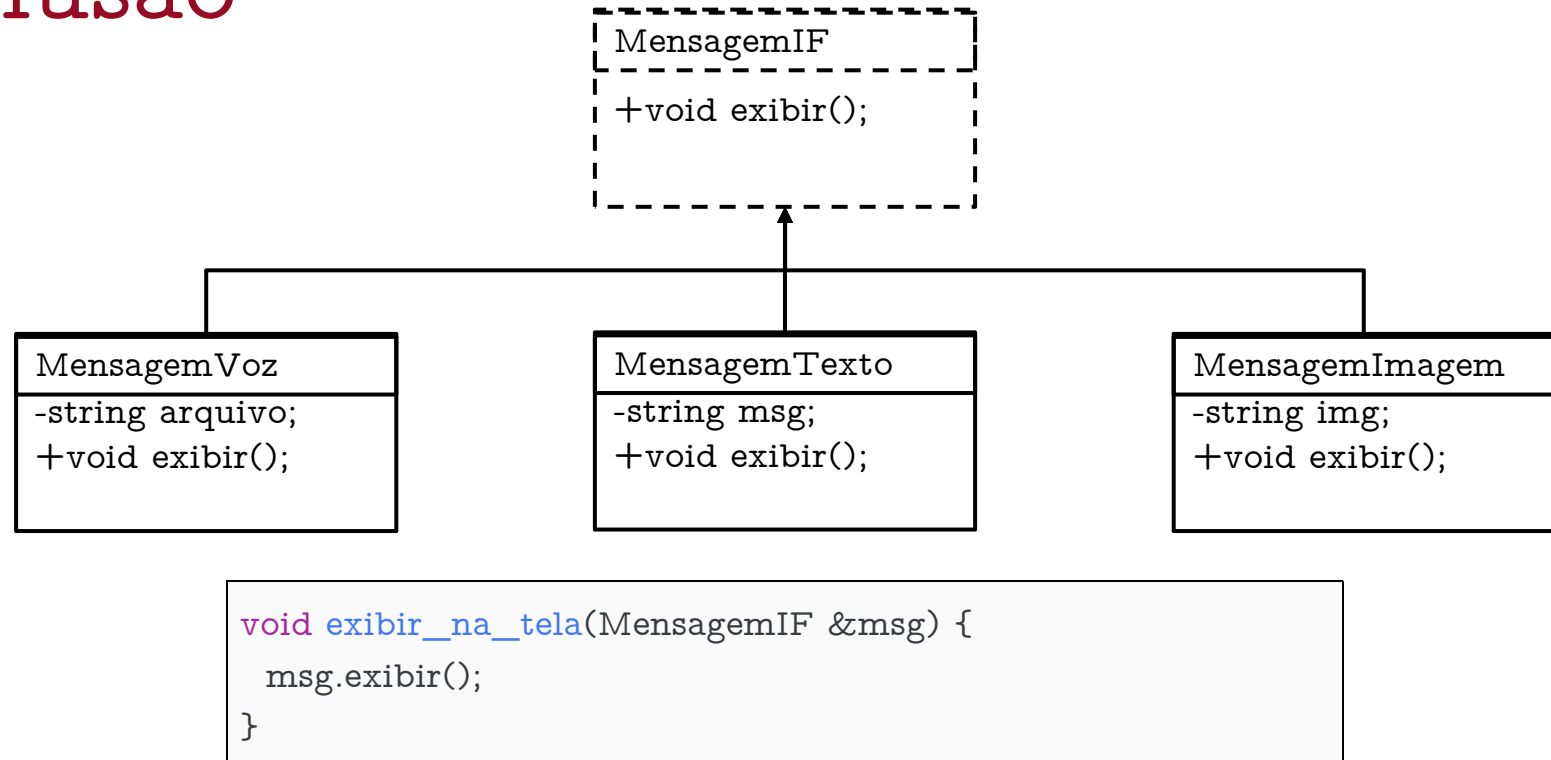


- Todas suportam o `exibir`



# Tipos de polimorfismo

## Inclusão

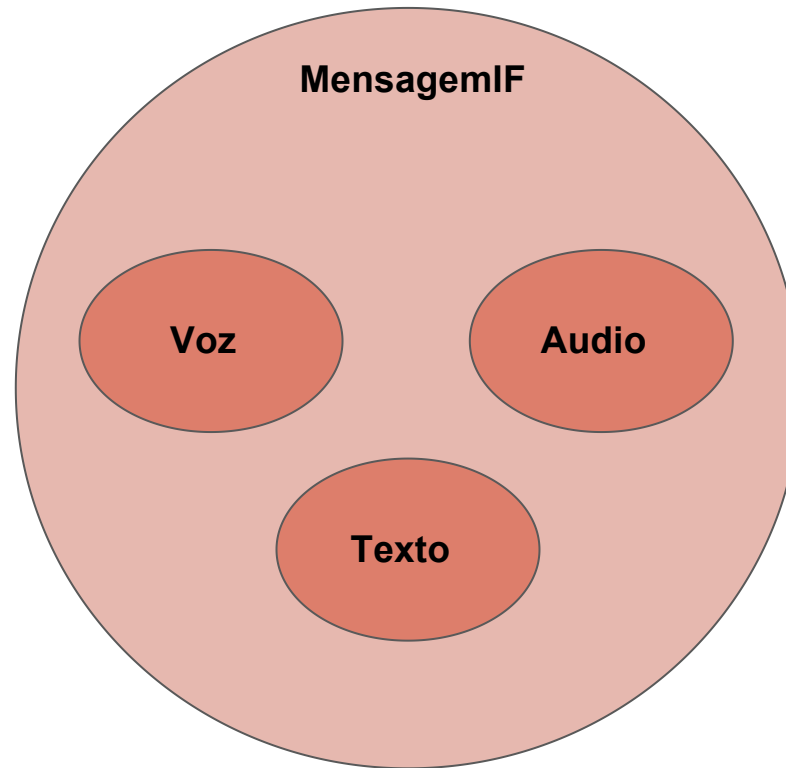


- Porém cada uma tem comportamento interno (private) diferente

# Tipos de polimorfismo

## Inclusão

### Contexto de Tipos



# Tipos de polimorfismo

## Ad-hoc - Sobrecarga

- Número finito de entidades distintas, todas com mesmo nome, mas códigos distintos
- Função ou valor conforme o contexto

# Tipos de polimorfismo

## Ad-hoc - Sobrecarga

- O mesmo identificador denota diferentes funções que operam sobre tipos distintos
- Resolvido estaticamente (compilação)
  - Considera os tipos para escolher a definição
  - Difere no número e no tipo dos parâmetros

# Tipos de polimorfismo

## Ad-hoc - Sobrecarga

```
class Ponto {
private:
    double __x = 0;
    double __y = 0;
public:
    void set_xy(double x, double y) {
        this->__x = x;
        this->__y = y;
    }

    void set_xy(double xy) {
        this->__x = xy;
        this->__y = xy;
    }
};
```

# Conversão de tipos (Casting)

---

# Conversão de tipos

- Uma classe, ao herdar de outra, assume o tipo desta onde quer que seja necessário
- Upcasting
  - Conversão para uma classe mais genérica
- Downcasting
  - Conversão para uma classe mais específica

# Conversão de tipos

## Upcasting

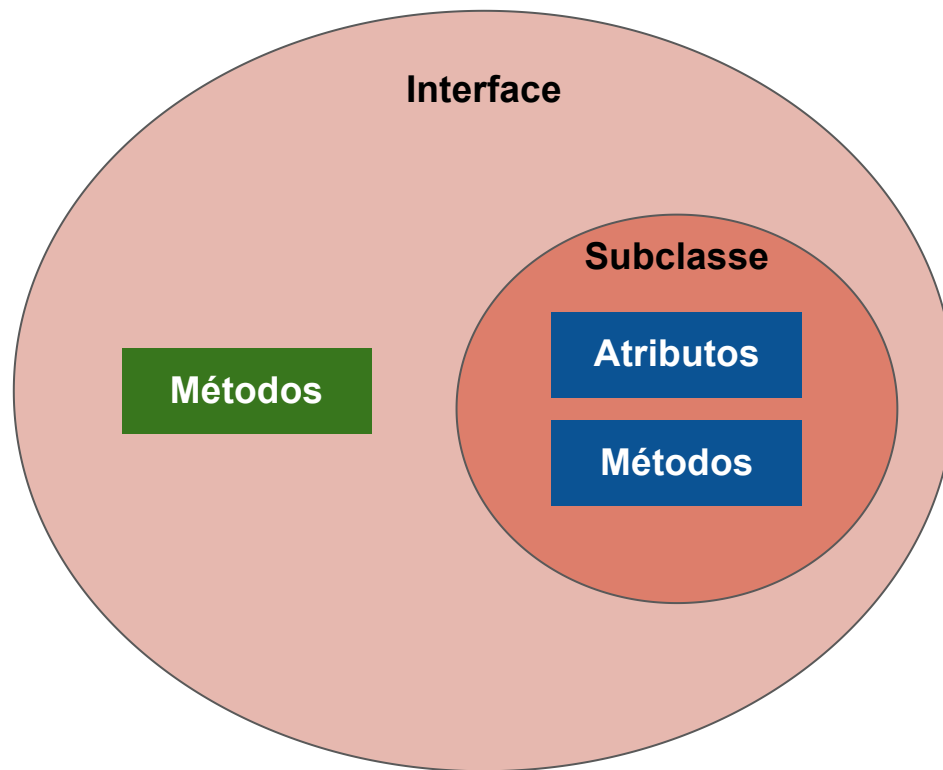
- Ocorre no sentido Classe  $\Rightarrow$  Interface
- Não há necessidade de indicação explícita
- A classe derivada sempre vai manter as características públicas da superclasse



# Conversão de tipos

## Upcasting

### Contexto de Classe



# Upcasting

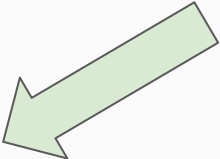
Note que os objetos viram MensagemIF auto-magicamente

```
#include "mensagem.h"
#include "mensagemimg.h"
#include "mensagemtexto.h"
#include "mensagemvoz.h"

void exibir_na_tela(MensagemIF &msg) {
    msg.exibir();
}

int main(void) {
    MensagemTexto texto("Oi, tem aula de PDS2 hoje?");
    MensagemVoz audio("audio.wav");
    MensagemImagem image("imagem03.ascii");
    MensagemTexto texto2("Mas que puxa :)");

    exibir_na_tela(texto);
    exibir_na_tela(audio);
    exibir_na_tela(image);
    exibir_na_tela(texto2);
}
```



# Conversão de tipos

## Downcasting

- Ocorre no sentido Interface  $\Rightarrow$  Classe
- Não é feito de forma automática!
- Deve-se deixar explícito, informando o nome do subtipo antes do nome da variável

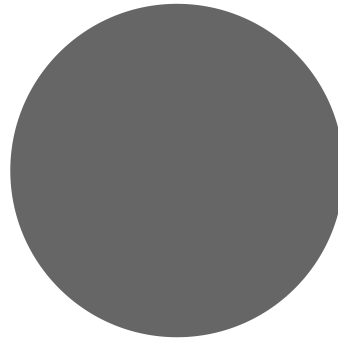
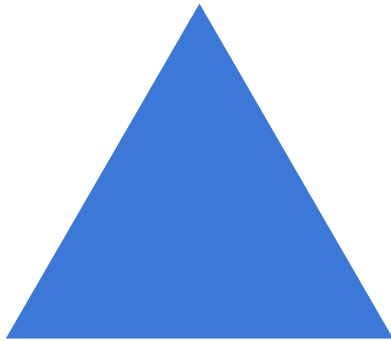
# Conversão de tipos

## Downcasting

- Nem sempre uma superclasse poderá assumir o tipo de uma subclasse
- Toda MensagemTexto é uma MensagemIF
- Nem toda MensagemIF é MensagemTexto
- Caso não seja possível
  - Segmentation fault

# Exercício

- Como modelar os elementos abaixo?
- Quais dados possuem em comum?
- Quais operações podem ser aplicadas?



# Exercício

- Análise inicial
  - Figuras geométricas
  - Dados: Cor
  - Operações: Área e Perímetro

